Klausur Datenbanken
Wintersemester 2007/2008
Prof. Dr. Wolfgang May
5. Februar 2008, 11-13 Uhr
Bearbeitungszeit: 90 Minuten

Vorname:

Nachname:

Matrikelnummer:

Studiengang:

Bei der Klausur sind **keine Hilfsmittel** (Skripten, Taschenrechner, etc.) erlaubt. Handies müssen ausgeschaltet sein. Papier wird gestellt. Benutzen Sie nur die **ausgeteilten**, zusammengehefteten **Blätter** für Ihre Antworten. Schreiben Sie mit blauem/schwarzem Kugelschreiber, Füller, etc.; Bleistift ist nicht erlaubt.

Auf dem letzten Blatt finden Sie eine Datenbasis, die in den Aufgaben 1 und 2 verwendet wird. Trennen Sie es ggf. zur Bearbeitung der Aufgaben ab.

Zum **Bestehen** der Klausur sind **45** Punkte hinreichend.

☐ meine Note soll mit Matrikelnummer so bald wie möglich auf der Vorlesungs-Webseite veröffentlicht werden.

☐ meine Note soll nicht veröffentlicht werden; ich erfahre sie dann aus Munopag/Wopag/FlexNever oder beim zuständigen Prüfungsamt.

	Max. Punkte	Schätzung für "4"
Aufgabe 1 (ER-Modell)	15	10
Aufgabe 2 (SQL und Relationale Algebra)	35	15
Aufgabe 3 (Anfrageoptimierung und Auswertung)	21	7
Aufgabe 4 (Transaktionen)	19	12
Summe	90	44

Note:

Aufgabe 1 (ER-Modell [15 Punkte])

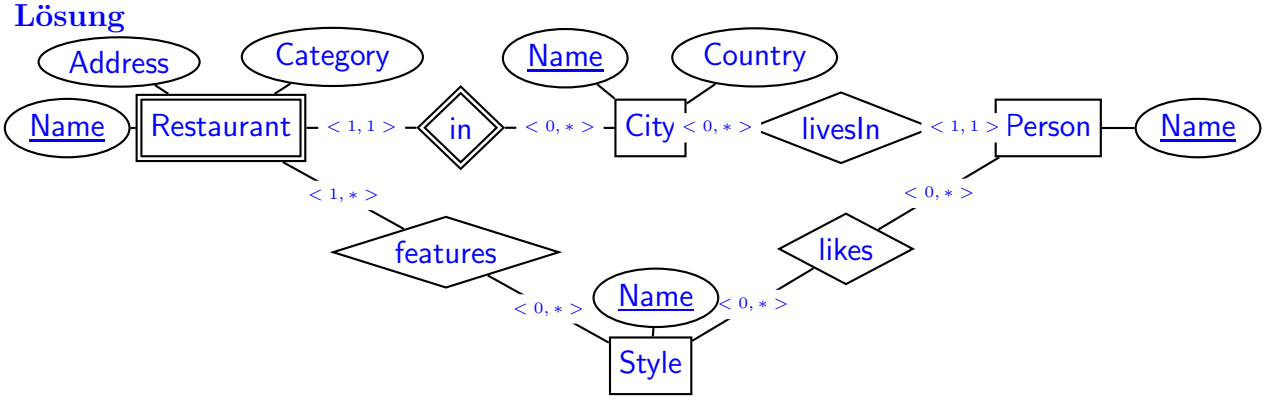
Geben Sie ein ER-Modell für den folgenden Sachverhalt an (die für die SQL-/Algebra-Aufgabe angegebene Datenbasis ist ein Teil dieses Szenarios). Geben Sie Schlüssel und Kardinalitäten sinnvoll an.

- Es geht um eine Datenbank über Restaurants.
- Jedes Restaurant hat einen Namen und befindet sich in einer Stadt (Hinweis: für Städte wird der Name als Schlüssel angenommen). In verschiedenen Städten kann es Restaurants mit demselben Namen geben ("Ratskeller", "Zum Goldenen Ochsen", "La Trattoria"). Jedes Restaurant hat eine Adresse.
- Jedem Restaurant ist eine Preiskategorie zugeordnet (extrem teuer=5, billig=1).
- Für jedes Restaurant ist angegeben, welche Art von Gerichten dort angeboten wird (Italienisch, Fisch, ...). Ein Restaurant kann dabei mehrere Einträge haben (z.B. passen Indisch und Vegetarisch gut zusammen). "Local" bedeutet, dass dort die landesübliche Küche gepflegt wird.

Weiterhin gibt es Personen:

- Jede Person hat einen Namen und lebt in einer Stadt.
- Jede Person bevorzugt ein oder mehrere Koch- bzw. Küchenstile (z.B. Fisch, Portugiesisch und French Cuisine).

Lösung



- Restaurant mit Name als Schlüssel ist ein schwacher Entitätstyp, da es (siehe Text) mehrere Restaurants desselben Namens in verschiedenen Städten geben kann (-1P).
- Alternative: Restaurant als starker Entitytyp wenn die Adresse (auch) Schlüsselattribut ist. Dies entspricht allerdings dann nicht den angegebenen Tabellen: aus Style(City, Restaurant, Style) kann man erkennen, dass Style.(Restaurant,City) als Fremdschluessel Restaurant(Name,City) referenziert.
- Kardinalitäten bei City jeweils <0,1>; sonst kann man keine Restaurants abspeichern, die in Städten liegen, in denen man keine Personen gespeichert hat (und umgekehrt; -0.5P).
- "City.Country" stand nicht im Text, aber in den Tabellen (+0.5P).

Aufgabe 2 (SQL und Relationale Algebra [35 Punkte])
Verwenden Sie für diese Aufgabe die Datenbasis, die auf dem letzten Blatt der Klausur angegeben ist.

a) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck bzw. Baum an, der die Menge der Namen aller Städte ergibt, in denen man italienisch essen kann. (3P)

Lösung

```
SELECT DISTINCT City
FROM Styles
WHERE Style='Italian';
```

π[City](σ[Style='Italian'](Styles))

b) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck bzw. Baum an, der die Namen aller Personen ergibt, die kein italienisches Essen mögen. (6P)

Lösung

```
(SELECT Name
 FROM Person)
MINUS
(SELECT Name
 FROM likes
 WHERE Style = 'Italian')
```

```
SELECT Name
FROM Person
WHERE NOT EXISTS
(SELECT *
 FROM likes
 WHERE Style = 'Italian'
 AND Name = Person.Name)
```

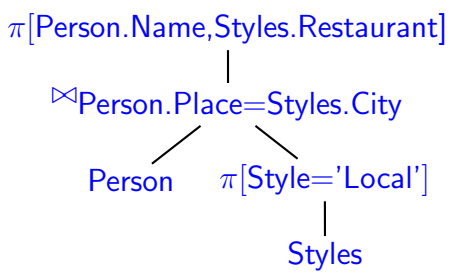
```
SELECT Name
FROM Person
WHERE Name NOT IN
(SELECT Name
 FROM likes
 WHERE Style = 'Italian')
```

π[Name](Person) $-$ π[Name](σ[Style='Italian'](likes))

c) Geben Sie eine SQL-Anfrage *und* einen Algebra-Ausdruck bzw. Baum an, die für jede Person angibt, in welchen Restaurants ihrer Heimatstadt sie landesübliche Gerichte (Style="Local" bedeutet, dass ein Restaurant landesübliche Küche anbietet) angeboten bekommt [Antworten als Paare (Person, Restaurant)]. (6P)

Lösung

```
SELECT Person.Name, Styles.Restaurant
FROM Person, Styles
WHERE Person.Place = Styles.City and
      Styles.Style = 'Local';
```



d) Geben Sie eine eine SQL-Anfrage *oder* einen Algebra-Ausdruck bzw. Baum an, die die Menge der Namen aller Restaurants ergibt, in denen Alice und Peter sich zum gemeinsamen Essen verabreden können, und beide etwas finden, das sie mögen. (6P)

Lösung Wichtig ist es, dabei zu berücksichtigen, dass Alice und Peter nicht unbedingt denselben Stil mögen, sondern dass die in Frage kommenden Restaurants eventuell mehrere Stile anbieten.

Dies war die Aufgabe mit den meisten grundverschiedenen Lösungen

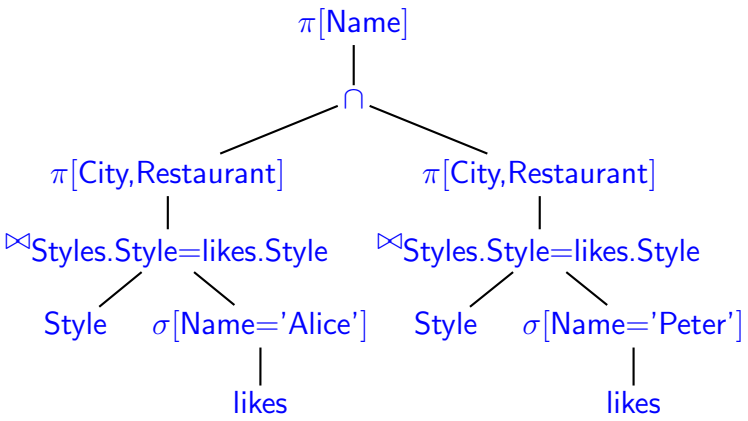
Sehr explizite "Musterlösung":

```
SELECT DISTINCT Name
FROM Restaurant A
WHERE EXISTS
(SELECT *
 FROM Styles B
 WHERE A.City = B.City
   AND A.Name = B.Restaurant
   AND B.Style in
    (SELECT Style
     FROM likes
     WHERE Name='Alice'))
AND EXISTS
(SELECT *
 FROM Styles B
 WHERE A.City = B.City
   AND A.Name = B.Restaurant
   AND B.Style in
    (SELECT Style
     FROM likes
     WHERE Name='Peter'))
```

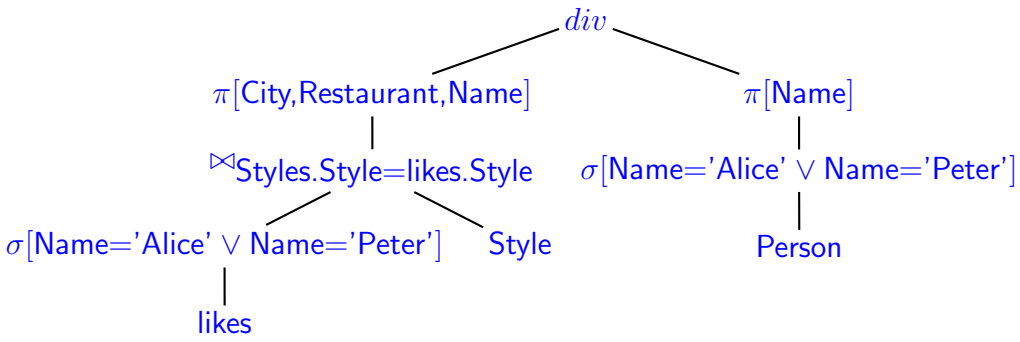
Kürzer und breiter:

```
SELECT r1.name
FROM likes l1, styles s1, styles s2, likes s2
WHERE l1.name = 'Alice'
  AND l1.style = s1.style
  AND s1.restaurant = s2.restaurant
  AND s1.city = s2.city
  AND s2.style = l2.style
  AND l2.name = 'Peter'

SELECT restaurant
FROM ((SELECT restaurant, city
       FROM likes, styles
       WHERE likes.name = 'Alice'
         AND likes.style = styles.style)
      INTERSECT
      (SELECT restaurant, city
       FROM likes, styles
       WHERE likes.name = 'Peter'
       AND likes.style = styles.style))
```



Hinweis: anstelle von ∩ ist auch ⋈ korrekt.



Hinweis: im linken Teilbaum kann man die Selektion auf Alice und Peter sogar weglassen – die Division prüft nur, ob Alice und Peter beide in das jeweilige Restaurant gehen würden.

Die Divisions-Lösung würde es allgemein ermöglichen, im rechten Teilbaum eine beliebige Personengruppe zu selektieren, und nach einem Restaurant zu suchen, in dem sich diese Personen treffen können.

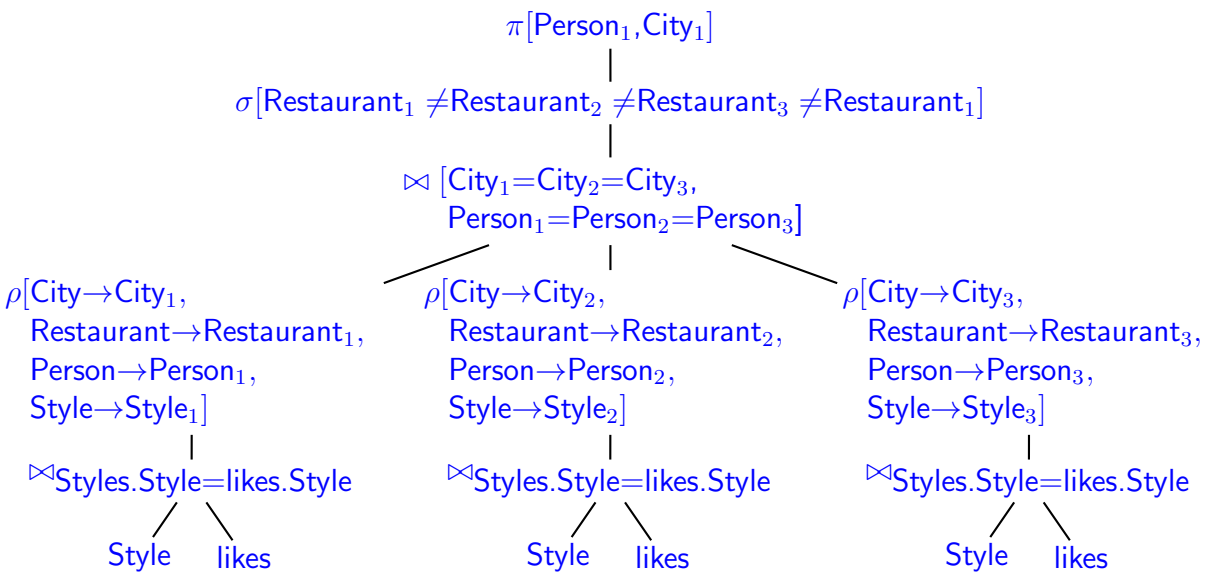
e) Geben Sie eine SQL-Anfrage an, die für jede Stadt und Person die Anzahl der Restaurants ergibt, in denen diese Person gerne essen gehen würde. Es sollen nur Städte ausgegeben werden, in denen es mindestens drei solche Restaurants gibt. (7P)

Lösung

```
SELECT Person.Name, Restaurant.City, count(Restaurant.Name)
FROM Person, Restaurant
WHERE EXISTS
  (SELECT *
   FROM likes, Styles
   WHERE likes.Name = Person.Name
     AND likes.Style = Styles.Style
     AND Styles.Restaurant = Restaurant.Name
     AND Styles.City = Restaurant.City)
GROUP BY Restaurant.City, Person.Name
HAVING Count (Restaurant.Name) >= 3.

SELECT likes.Name, Styles.City, count(DISTINCT Restaurant.Name)
FROM likes, Styles
WHERE likes.Style = Styles.Style
GROUP BY likes.Name, Styles.City
HAVING Count (DISTINCT Restaurant.Name) >= 3.
```

Bemerkung: Man in diesem Fall die Aggregation (Count) auch durch ein Join ersetzen und dann auch als Algebra-Ausdruck formulieren:



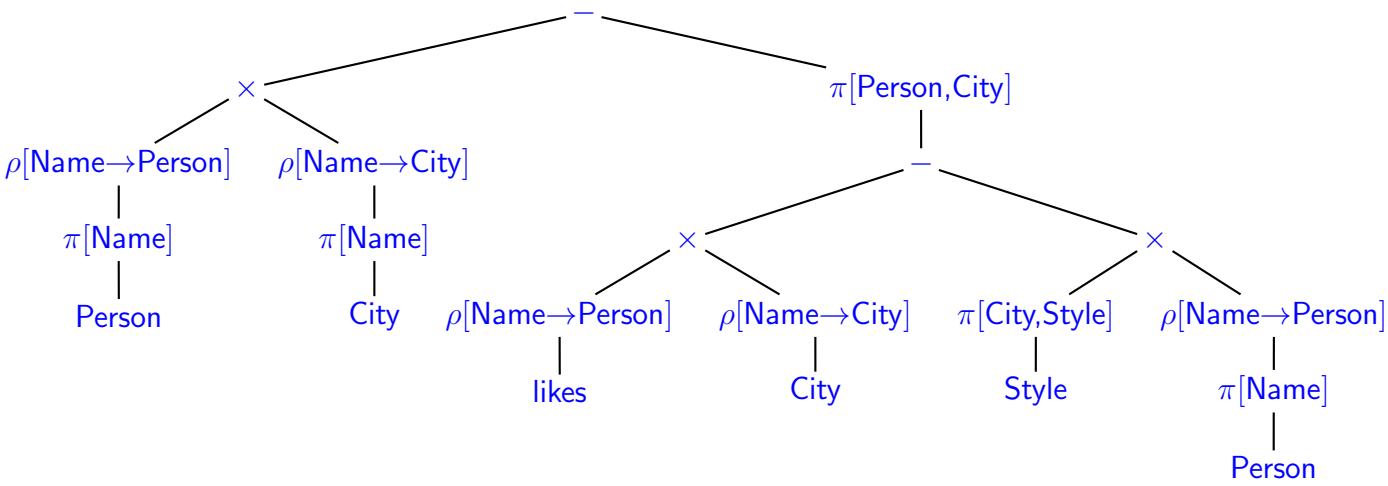
f) Geben Sie eine SQL-Anfrage *oder* einen Algebra-Baum *oder* einen Algebra-Ausdruck an, der alle Paare (Person, Stadt) zurückgibt, so dass jeder der von dieser Person bevorzugten Stile in dieser Stadt durch mindestens ein Restaurant vertreten ist. (7P)

Lösung

```
SELECT Person.Name, City.Name
FROM Person, City
WHERE NOT EXISTS
  -- preference of that person that is not satisfied by any
  -- restaurant in this city
(SELECT Style
 FROM likes
 WHERE Name = Person.Name
   AND NOT EXISTS
       (SELECT *
        FROM Styles
        WHERE City = City.Name
          AND Style = likes.Style))
```

In diesem Fall ist der relationale Ausdruck komplizierter, da es sich um eine *korrelierte Division* handelt. Man müsste "blockweise" das Angebot der Städte (City x Style) für jede Person durch ihre "likes" dividieren.

Die folgende Lösung stellt genau die obige SQL-Anfrage mit doppeltem Minus nach:



- Result of the middle $\times$: all tuples (Person,Style,City) such that person p likes/requires style s in all cities, thus in every city c.
- Result of the right $\times$: all tuples (City,Style,Person) such that style s is available in city c, thus, available for all persons p.
- right "-": all tuples (p, s, c) such that style s is liked/required by person p, but not offered in city c.
- right π: the resulting pairs (p, c) such that person p misses some style in city c.
- left $\times$: all pairs (person,city).
- upper "-": those pairs (person,city) that person p does not miss any liked style in city c.

Aufgabe 3 (Anfrageoptimierung und Auswertung [21 Punkte])
Gegeben sind zwei Tabellenschemata R und S:

R		
A	B	C

S	
E	F

Die Aufgabe behandelt das Join $R \bowtie_{R.A=S.E} S$.
Physikalisches Datenbankschema:

- Schlüssel von R ist (A, B), Schlüssel von S ist (E, F).
- Der Datentyp von $R.A$ und $S.E$ ist "Integer", d.h., ganze nichtnegative Zahlen.
- Der Datentyp von $R.C$ ist CHAR(20), d.h. Zeichenketten der festen Länge 20.
- Die Tupel von R sind in beliebiger Reihenfolge gespeichert, es existieren keine Indexe auf R.
- Die Tupel von S sind in beliebiger Reihenfolge gespeichert, es existieren keine Indexe auf S.

Daten:

- R enthält 2.000.000 Tupel.
- S enthält 4.000.000 Tupel.
- Die Tabellen sind wie üblich seitenweise abgelegt. Jede Seite (4KB) von R enthält 50 Tupel, jede Seite von S enthält 100 Tupel.
- Im Hauptspeicher können 1020 Seiten gleichzeitig gehalten werden.
- 491, 907 und 1019 sind Primzahlen.

Fragen (jeweils mit kurzer Begründung zu beantworten):

a) Wieviele Speicherseiten umfassen R und S? (2P)

b) Wieviele Bytes umfasst (grob) ein R-Tupel? (1P)

c) Die Anfrage $\pi[A, B, F](R \bowtie_{R.A=S.E} S)$ soll berechnet und am Bildschirm ausgegeben werden.
Geben Sie den Algebra-Baum der Anfrage an und optimieren Sie die Anfrage algebraisch. (2P)

d) Beschreiben Sie, wie man diese Anfrage auswertet, wobei ein *Hash-Join* verwendet werden soll; Sie dürfen dazu bis zu 1000 Hauptspeicherseiten verwenden.
Wieviele Hintergrundspeicherzugriffe auf die Datenbank werden benötigt?
(beschreiben Sie am besten zuerst grob, wie Sie vorgehen, und analysieren Sie dann die Zugriffe) (ingesamt 12P)

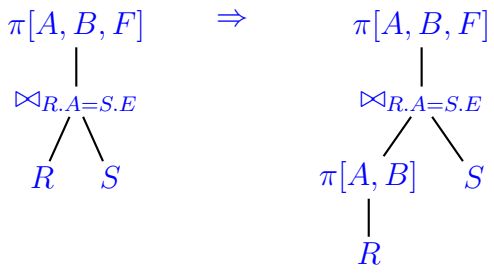
e) (Bearbeiten Sie diesen Teil erst, wenn Sie die Lösung für (d) ausgearbeitet haben!)
Annahme: man weiss, dass $\pi[A](R) \subsetneq \pi[E](S)$ ("echte Teilmenge") ist, d.h. für z.B. 60% der Werte von $S.E$ kein passender Wert in $R.A$ vorhanden ist. An welcher Stelle (und wie) kann man das obige Verfahren dann optimieren? (4P)

Lösung Bei der Aufgabe wird nicht erwartet, dass alles komplett richtig ist, sondern dass die Argumentation vernünftig ist.

a) R umfasst 2.000.000/50 = 40.000 Speicherseiten, S umfasst 4.000.000/100 = 40.000 Speicherseiten.

b) Eine Seite hat 4KB (4096 Bytes), 50 Tupel, also ist ein Tupel ca. 80 Bytes gross.

c) Da $R.C$ im Ergebnis nicht verwendet wird, kann man die Projektion $\pi[A, B](R)$ vor dem Join ausführen.



d) Grober Ablauf: R durchgehen, nach $R.A$ hashen (dabei kann man $R.C$ gleich wegprojizieren), S durchgehen, mit selber Funktion nach $S.E$ hashen. Dann "gegenüberliegende" Hash-Partitionen (d.h. gleicher Hashwert) miteinander im Hauptspeicher verknuepfen.

Man benötigt keine Duplikateliminierung, da (A, B) Schlüssel von R ist und $(A = E, F)$ Schlüssel von S ist und somit keine Duplikate bei der Auswertung entstehen können.

Detailbeschreibung. Man hat 1020 Seiten, kann also bis zu 1019 Hash-Partitionen bilden. (mit jeder Zahl zwischen ca. 100 und 1019 geht es auch). Man wählt also z.B. $R.A/491$ als Hash-Funktion.

R hashen: 40000 Zugriffe, verteilen, und wenn eine Hash-Seite voll ist, in der Hintergrundspeicher schreiben. Maximal also 40000+40000+491 Zugriffe (+491 da nicht alle Seiten exakt gefüllt werden).

Wenn man an dieser Stelle bereits $R.C$ (20 Zeichen) wegprojiziert, benötigt jedes Tupel nur noch 60 statt 80 Bytes, man kommt also mit ca. 30000+491 neuen Seiten aus.

S hashen: 40000 Zugriffe, verteilen, und wenn eine Hash-Seite voll ist, in der Hintergrundspeicher schreiben. Maximal also 40000+40000+491 Zugriffe.

Überschlagsrechnung:
Pro Hashwert hat man in R etwa 2.000.000/491 = 4073 Tupel, also 81 Seiten (Wenn man $R.C$ wegprojiziert hat, sind es noch weniger, s.u.). In S hat man pro Hashwert etwa 4.000.000/491 = 8146 Tupel, also ebenfalls 81 Seiten. Man kann also davon ausgehen, dass die Verknüpfung gegenüberliegender Hashpartitionen im Hauptspeicher stattfinden kann.

(Nebenbemerkung: also 100 Partitionen hat, hat jede Partition von R 1/100 der Seiten von R, also 400. Genauso für S. Um gegenüberliegende Partitionen im Hauptspeicher verknüpfen zu können benötigt man also durchschnittlich 800 Seiten – 1020 hat man. Bei ungleicher Verteilung wird es dann so langsam knapp. Man sollte also mindestens 100 Partitionen bilden. Mit 491 wie oben ist man auf der sicheren Seite und "fair" indem man nur 50% des Hauptspeichers belegt. Wie sich an der Rechnung zeigt, wird bei mehr Partitionen die Anzahl der Hauptspeicherzugriffe sogar ein wenig höher.)

Also alles nochmal lesen: 40491 + 40491 = 80892 Zugriffe, jedes Ergebnis wird unmittelbar ausgegeben. Gesamt: 241964 Zugriffe.

Wenn man $R.C$ gleich wegprojiziert:

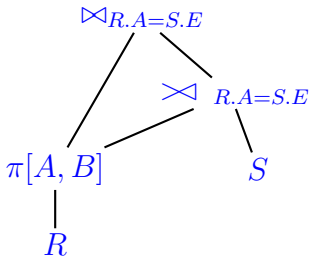
R hashen: 40000 + 30500
S hashen: 40000 + 40500
alles nochmal lesen: 30500+40500 = 71000
Summe: 222000.

e) Man muss in die Hashpartitionen für S garnicht jedes Tupel eintragen, sondern nur diejenigen, die ein Gegenstück in R haben. Dazu kann man während man R hasht in den noch freigelassenen 20 Hauptspeicherseiten einen Baumindex über $R.A$ (Integer, also wenig Platzbedarf) aufbauen, mit dem man jedes S-Tupel bei Bearbeitung abgleicht.

Man erreicht damit (Annahme dass 60% der Werte von $S.E$ in $R.A$ nicht vorkommen) eine Reduktion der S-Tupel:

R hashen: 40000 + 30500
S hashen: 40000 + 16500
alles nochmal lesen: 30500+16500 = 47000
Summe: 174000

Anmerkung: Diese Auswertung kann man auch im Algebra-Baum (der damit zum azyklischen Graphen wird) mit einem Semijoin repräsentieren:



Nebenbemerkung. Bei dieser Aufgabe wurde häufig *Hash-Join* mit einem Nested-Loop-Index mit Benutzung eines *Hash-Indexes* verwechselt. Ein Hash-Index bringt hier relativ wenig: Hash-Index auf $S.E$, dann R als äußere Relation (R ist kleiner). Erstmal muss man den Hash-Index auf $S.E$ erstellen. $S.E$ ist Integer, also braucht wenigstens mal nicht viel Platz.
Ziel: jede Kachel muss alle "ihre" Tupel, die irgendwo in den Seiten von S liegen, referenzieren. Man wählt die Hash-Funktion möglichst gross, um weniger Einträge pro Kachel zu haben. Beim Aufbauen des Hash-Indexes ist jedoch zu bedenken, dass man für jeden Hash-Wert eine Kachel, die man gerade füllt, im Hauptspeicher haben sollte. Also, maximal 1019 Kacheln. 1019 ist eine Primzahl, also nimmt man n mod 1019.
Sinnvollerweise speichert man jetzt in jeder Kachel nicht nur die Referenzen, sondern auch gleich die Werte von $S.E$. Für jeden Eintrag muss man die Seite (1..40000 $\rightarrow$ 2 Byte) und die Position (1..50 $\rightarrow$ 1 Byte) und den Wert von $S.E$ (in 4 Byte) ablegen, ist also 7 Byte gross. Pro Seite kann man dann 570 Einträge ablegen (Wenn man pro Seite Gruppen der Form (Wert, Ref_1, ..., Ref_n) anlegt, können es sogar mehr sein) . Man benötigt also maximal 4.000.000/570+1019=7017+1019 Seiten (weiterrechnen mit 8000). Jede Hash-Kachel umfasst ca. 8 Seiten.
Erstellen des Hashes: alle Seiten linear durchgehen, Hash-Index aufbauen: 40000 lesen, 8000 schreiben = 48000.
Und dann den Nested Loop-Join durchführen: R seitenweise laden, für jedes Tupel $hash(R.A)$ berechnen, Index-Kachel-Seiten holen. Es passt nur $1/8$ des Index in den Speicher, man muss also in 87.5% der Fälle eine Kachel aus dem Hintergrund holen. Dann darin (lokal) suchen, welche Werte wirklich passen (es werden ja immer mehrere $S.E$ auf denselben Wert gehasht),

und *jedes* passende S-Tupel aus der S-Seite des Hintergrundspeichers holen und das Ergebnis ausgeben.
Jetzt kommt es natürlich sehr auf die Selektivität an - Annahme: es gibt zu jedem Tupel aus S k passende Tupel aus R (die für kleines k üblicherweise auf k verschiedenen Seiten liegen): 40000 Seiten $\cdot 50$ Tupel $\cdot$ ((Wahrscheinlichkeit $0.875 \cdot 8$ Index-Seiten $) + (k$ R-Tupel $)) = 2.000.000 \cdot (k + 7)$
was offensichtlich sehr viel schlechter ist, als der Hash-Join.
Ein Hash-Index (wie auch ein B$*$-Baum-Index) ist gut, um auf *ein* (oder wenige k) Tupel gezielt zuzugreifen. Als Join-Algorithmus taugt es aber nicht. Der Vorteil des Hash-Joins liegt darin, dass es die Tupel *effizient* so umordnet dass man nachher zusammenpassende Tupel auch zusammen findet.
Dies ist ein Paradebeispiel von "amortisierter Analyse" (Info III?): erst mehr investieren (alles einmal hashen) um nachher billiger wegzukommen.

Aufgabe 4 (Transaktionen [19 Punkte])
Gegeben ist der folgende Schedule S:

$$R_2A\ R_3A\ R_3B\ W_3C\ R_1C\ R_1B\ W_1B\ R_2B\ W_2A\ W_1C$$

a) Geben Sie den D-Graphen von S an (4P).

b) Geben Sie den C-Graphen von S an (4P).

c) Ist S serialisierbar (mit Begründung; 2P)

d) Betrachten Sie den *leicht veränderten* Schedule S', bei dem die beiden angegebenen Aktionen vertauscht sind:

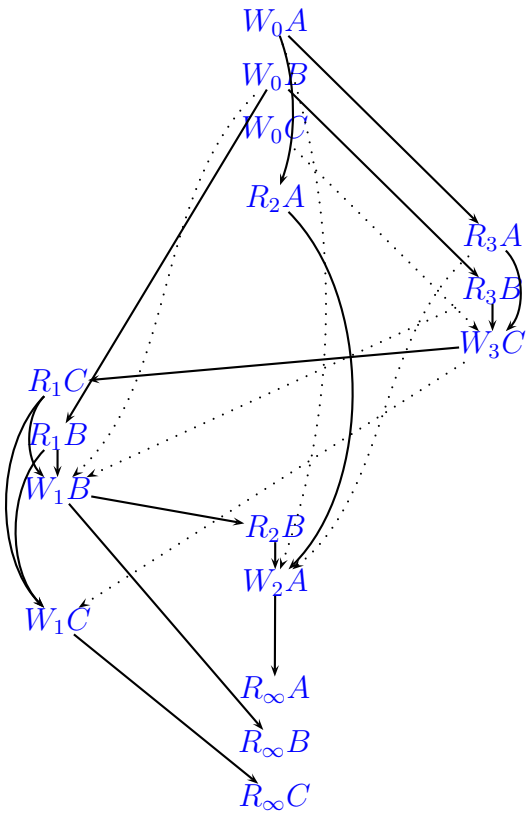
$$R_2A\ R_3A\ R_3B\ W_3C\ R_1C\ R_1B\ \boxed{R_2B\ W_1B}\ W_2A\ W_1C$$

Ist S' äquivalent zu S?
Ist S' serialisierbar?
(jeweils mit Begründung; 3+3P)

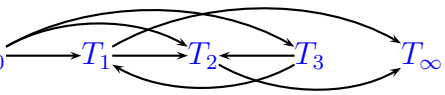
e) Ist S bzgl. einem Scheduler, der nach dem Zeitstempelverfahren arbeitet, zulässig (3P)?

Lösung

a) Dependency-Graph (weitere RW- und WW-Konflikte gestrichelt eingetragen)



b) Konfliktgraph:



c) Der Konfliktgraph ist zyklenfrei, also ist S serialisierbar; S ist äquivalent zu $T_3\ T_1\ T_2$.

Hinweis: das "blind write" W_3C hat keinen Einfluß auf die Aussage. Nur für die Umkehrung "Wenn der CG zyklisch ist und es keine blind writes gibt, dann ist S nicht serialisierbar" wäre dies relevant (siehe Vorlesung!).

d) S' ist nicht äquivalent zu S, da R_2B in S einen anderen Wert (den von T_1 geschriebenen) als in S' (den von T_0 geschriebenen) liest.

S' ist serialisierbar; S' ist äquivalent zu $T_3\ T_2\ T_1$. Vertauschen der beiden Aktionen W_1B und R_2B dreht den *einzigen* Konflikt zwischen T_1 und T_2.

e) Nein. T_2 bekommt den kleinsten Zeitstempel, 1. In der ersten Aktion setzt T_2 den Zeitstempel von A auf 1. Als nächstes beginnt T_3, Zeitstempel 2. R_3A ist zulässig und setzt den Zeitstempel von A auf 2.

An dieser Stelle kann man schon sehen, dass T_2 verloren ist, denn es darf später W_2A nicht mehr ausführen. Verfolgt man den Schedule, trifft man früher aber bereits auf W_1B und R_2B, wobei bei R_2B $T(B) = T(1) = 3 > T(2)$ ist.

[Trennen Sie dieses Blatt am besten vor Beginn der Bearbeitung ab]

Die folgende Datenbasis wird in Aufgabe 2 verwendet.

Restaurant

City	Name	Address	Category
Paris	L'Arpege	...	5
Paris	Buddha Bar	...	5
Paris	Le Train Bleu	...	4
Paris	Nioullaville	...	3
München	Il Grappolo	...	3
München	Makassar	...	3
München	Shoya	...	3
München	Augustiner	...	2
Vienna	Rimini	...	4
Vienna	Churrascaria	...	3
Lisbon	Sette Mares	...	3
Lisbon	Bica do Sapata	...	4
Lisbon	Estoril Mandarin	...	5
Lisbon	Armazem de Cachaca	...	4
Lisbon	Casanostra	...	3
Lisbon	Os Tibetanos	...	3
Roma	Gino da Trastevere	...	5
Roma	Agata e Romeo	...	4
Roma	Le Relais de Jardin	...	5
Roma	Margutta Vegetariano	...	4
:	:		

Styles

City	Restaurant	Style
Paris	L'Arpege	French
Paris	Buddha Bar	Indian
Paris	Le Train Bleu	French
Paris	Le Train Bleu	Local
Paris	Nioullaville	Chinese
München	Il Grappolo	Italian
München	Makassar	Indian
München	Makassar	Fish
München	Shoya	Japanese
München	Augustiner	Local
Vienna	Rimini	Italian
Vienna	Churrascaria	Brazilian
Lisbon	Sette Mares	Fish
Lisbon	Bica do Sapata	Portuguese
Lisbon	Bica do Sapata	Local
Lisbon	Estoril Mandarin	Chinese
Lisbon	Armazem de Cachaca	Brazilian
Lisbon	Casanostra	Italian
Lisbon	Os Tibetanos	Vegetarian
Roma	Gino da Trastevere	Fish
Roma	Gino da Trastevere	Local
Roma	Agata e Romeo	Fish
Roma	Agata e Romeo	Italian
Roma	Le Relais de Jardin	French
Roma	Margutta Vegetariano	Vegetarian
:	:	:

City

Name	Country
Lisbon	P
Paris	F
Rome	I
:	:

Person

Name	Place
Alice	Paris
Peter	Vienna
:	:

likes

Name	Style
Alice	French
Alice	Vegetarian
Alice	Indian
Alice	Fish
Alice	Local
Peter	Italian
Peter	Portuguese
Peter	Brazilian
Peter	Local
:	:

* information taken from several Marco Polo guides from the last decade.